Schedule

Week	Tuesday	Thursday
Week 1 (09/03-9/07)	1, 2	3
Week 2 (09/10-09/14)	4.1-4.3	4.4-4.5
Week 3 (09/17-09/21)	5.1	5.2
Week 4 (09/24-09/28)	5.3-5.7	JFLAP
Week 5 (10/01-10/05)	Midterm	6.1, 6.2, 6.4, 6.5
Week 6 (10/08-10/12)	7	7, 8
Week 7 (10/15-10/19)	Break	8
Week 8 (10/22-10/26)	9	9
Week 9 (10/29-11/02)	9	10
Week 10 (11/05-11/09)	Midterm	10, 11
Week 11 (11/12-11/16)	11	12
Week 12 (11/19-11/23)	12	Thanksgiving
Week 13 (11/26-11/30)	13	13
Week 14 (12/03-12/07)	14	14

Week 1 (09/03-9/07)

Day 1

Goals
- Understand the different kinds of problems we would investigate, with Python examples.
- Understand what "Python programs" we will consider in this class.
- Practice with string representations of standard kinds of inputs.

Chapter 1
- Introduction to Theory of Computation
- Different kinds of problems

Activity 1
- Discuss examples of tractable/intractable/uncomputable problems.
- Activity Handout[1]

Chapter 2
- What is a SISO Python program?
- Input and Output of a program.
- Practice programs (pseudocode sufficient, work at the board):
  - Takes a string input and runs forever (at least on some inputs).
  - Takes a string input and throws an exception (at least on some inputs).
  - Given a string representing a list of numbers separated by spaces, like "23 45 13", add the numbers up and return the sum as a string.
- Exercise 2.5 a-d (work at the computers).

[1] activities/activity1.html

- When are SISO programs equivalent? Examples of different but equivalent programs.
- Decision programs.
- Exercise 2.7 (talk with neighbor)

Activity 2
- How can a SISO program process the following kinds of inputs:
  - A list of an arbitrary number of integers.
  - Two strings of predetermined length (say each of length 20).
  - Two strings of non-predetermined length.
  - Three strings of non-predetermined length.
  - A list of an arbitrary number of strings of non-predetermined length.

Day 2

Goals
- Understand the fact that programs are simply strings and can be inputs to other programs.
- Understand the difference between running a program and reasoning about a program's execution.
- Understand the limitations of programming languages by showing there are programs that cannot exist.

Chapter 3
- Examples of decision programs: containsGAGA, yes, longerThan1K, maybeLoop.
- The countLines program; programs as input to other programs.
- Self-reflection: programs examining themselves (group work Figure 3.3).
- The program yesOnString. Try at various inputs (group work Figure 3.4).
- Python offers a command exec, which takes a Python file and executes it. Can we use it to implement the behavior described by yesOnString (group discussion)?
- The program yesOnSelf. (group work Figure 3.5).
- Write yesOnSelf using yesOnString as a helper.
- The program notYesOnSelf. Think through what happens when notYesOnSelf is run with input the earlier programs, as well as itself. (group work).
- Discussion of programs crashOnString, crashOnSelf, weirdCrashOnSelf.

Activity 3
- Discuss definitions of the outputs of the programs noOnString, noOnSelf.
- Are noOnSelf and notYesOnSelf equivalent programs, if they existed?
- Is a similar conclusion possible for noOnSelf as for notYesOnSelf?
- Exercise 3.3
- Activity Handout[2]

Homework 1 (Due 9/13 4pm in LYN 102/110) 3.2, 3.10, 3.11

[2] activities/activity3.html

Week 2 (09/10-09/14)

Day 1

Goals
- Review the basic methodology of proof by contradiction.
- Understand the terms Alphabet, Language, String, in the Theory of Computation context.

Chapter 3 recap
- Exercise 3.3
- Review of proofs by contradiction.
- Write a formal proof that the program yesOnString cannot exist.

Section 4.1
- Computational Problems and how they differ from programs
- Alphabets, Strings, (Formal) Languages
- Exercise 4.3 (group discussion)
- Various examples of languages and language constructions
  - Empty/All, containing empty string only, Python/Java programs
  - Strings accepted by a decision program
  - union/intersection/complement/concatenation/Kleene
- Activity 4, first two points

Activity 4
- Describe three formal languages that are sufficiently different from those listed in the book.
- Exercises 4.4, 4.6

Day 2

Goals
- Learn the basic terminology related to computational problems
- Understand what it formally means to solve a problem.
- Understand the difference between decidable and recognizable languages.

Sections 4.2-4.5
- Computational problems:
  - inputs, solution sets, positive and negative instances
  - Describe the SHORTESTPATH problem
  - Exercises 4.11, 4.13
  - Kinds of computational problems: search, optimization, threshold, function, decision
  - General computational problems vs decision problems. Figure 4.13
  - Exercise 4.15
- What does it mean to solve/compute/decide a computational problem? Provide both positive and negative examples.

- Computable vs uncomputable problems. Provide examples of uncomputable problems.
- The membership problem for a language. The language corresponding to a decision problem.
- Definition of decidable languages and decidable decision problems.
- Is the language of Java Programs decidable?
- Recognizable languages. Give examples of recognizable languages that are not decidable.

Activity 5
- Write a Python program that *decides* the language of all (finite) binary strings that contain at least two 1s. (Food for thought: does this program have to worry about non-binary strings?)
- Write a Python program that *decides* the language of all prime numbers.
- Write Python programs that *decide* the empty language and the language consisting of just the empty string, respectively.

Practice
- The problem GREATER takes as input a number, and has as solutions all numbers that are greater than this number. Is this a computational problem according to our definition?
- A *composite* number is a positive integer which has at least one non-trivial divisor. The problem FACTOR takes as input a number. If the number is composite, then the solution set is all of the number's non-trivial divisors. If the number is prime, the solution is "no" indicating no solution. This can be described as a "search" problem. Express it as a search problem by specifying a predicate function.
- Define "threshold", "optimization" and "decision" versions of the previous problem that have an analogous computational difficulty.
- Write a Python program that *decides* the language that contains all strings in the alphabets that have "matching parentheses", meaning that every "open parenthesis" is going to be matched by a "close parenthesis". The strings can contain any other letters inbetween.

Homework 2 4.12, 4.19, 4.20, 4.25, Bonus: 4.26

Week 3 (09/17-09/21)

Day 1

Goals
- Understand the definition of a Turing machine
- Understand the graphical representation of a Turing machine
- Be able to follow the execution of a Turing machine on a given input

Section 5.1
- Definition of a Turing machine: alphabet, states, transition functions

- Transition function can be thought of as a combination of: new state function, new symbol function, direction function
- What different final states can we have in a Turing machine? Are any of them required?
- What is the difference between *looping* and *halting*?
- What Turing machines do we call *transducers* and which do we call *accepters*? Are those the only kinds of Turing machines?
- Abbreviated notation for state diagrams.

Activity 6
- Write a Turing machine that searches for the first C that occurs in the string and changes it to a G, then halts. If it arrives at the end of the input without finding a C, then it inserts a C and halts.
- Our alphabet contains only the letters C, G, T, A and the special letter x. Write a Turing machine that inserts an x at the beginning of the input, shifting all other letters to the right, and also inserts an x at the end of the input.
- Consider the alphabet consisting of only the binary digits 0, 1. Consider the input as a number with the least significant digit being at the beginning of the tape. So if the number was 8, the tape would start with "001" with the first 0 being at the start of the tape. Write a Turing machine that would change the tape so that the final output is the number incremented by 1. So in the example above the output would be "101". And if we had started with "101" as input, then the output would be "011" etc.

Day 2

Goals
- Understand how a Turing machine can use the tape to remember an unbounded amount of information.
- Understand how a Turing machine can be used as a component/subprogram in another Turing machine.

Section 5.2
- A Turing machine that checks whether the string has more Cs than Gs. It follows two different paths depending on whether it encounters a G or a C first.
- Study the binaryIncrementer machine.
- Practice: Define and construct a binaryDecrementer machine.
- Understand how binaryIncrementer and shiftInteger come together to produce a incrementWithOverflow.
- Understand the construction of the countCs Turing machine.

Activity 7
- Modify the countCs Turing machine to count the occurence of CA pairs instead.
- Write a Turing machine compares two numbers as follows:

- The numbers are provided in binary form from highest significant number to lowest, separated by x's and with zeroes padded in if needed to make sure both numbers have the same number of digits. For example the numbers 6 and 2 would be represented as: "x110xx010x".
- Your machine is free to change the tape contents as it needs to. You may also introduce up to two new symbols: y and z.
- Your machine must terminate in one of three states signifying whether the first number is larger than, equal to, or smaller than the second number.

Homework 3 5.3, 5.5, 5.6, 5.7

Week 4 (09/24-09/28)

Day 1

Goals
- Understand how two-tape and multi-tape machines have the same power as one-tape machines.
- Understand how a "random access" Turing machine has the same power as our simple one-tape Turing machine.
- Understand how a Python program can simulate a Turing machine.

Sections 5.3-5.7
- A two-tape single-head Turing machine can be simulated by a standard Turing machine by using an alphabet consisting of pairs of characters from the alphabet of the original machine.
- A multi-tape single-head Turing machine can be simulated in the same way.
- A multi-tape multi-head Turing machine can be simulated by using a multi-tape single-head Turing machine, by using twice the number of tapes, and using the second set of tapes to keep track of the locations of the multiple heads.
- Describe how a *random-access* Turing machine operates, and how to simulate it with a multi-tape Turing machine.
- Describe how the parts of a modern computer map to elements of a random-access Turing machine.
- How can a Python program be used to simulate a Turing machine?

Activity 8
- Implement countCs as a two-tape two-head Turing machine.
- Implement countCs as a two-tape single-head Turing machine.
- Describe a two-tape Turing machine that contains a string in one tape and an integer in the other tape. The machine then must count that many spots into the string as indicated by the integer.

Day 2

Goals
- Practice the use of JFLAP to build and test Turing machines.

Activity 9
- Use JFLAP to build a single-tape Turing machine that appends a z at the end of the input.
- Use JFLAP to build a single-tape Turing machine that prepends a z at the beginning of the input and shifts everythign else to the right. Assume the alphabet consists of the characters x,y,z,0,1.

Homework 4 5.10, 5.13

Week 5 (10/01-10/05)

Day 1

– **Midterm 1**

Day 2

Goals
- Understand the idea of *universal computer programs* and *universal Turing machines*.
- Be able to explain how a universal Turing machine would function.
- Familiarize yourself with programs that alter other programs.
- Use a universal computer program to produce programs that are recognizable but not decidable.

Sections 6.1, 6.2, 6.4, 6.5 (6.3 optional)
- Describe instances you are familiar with of programs executing other programs.
- Understand the program universal.py and how it works.
- Describe how a universal Turing machine would operate.
- Describe the "simulate-and-alter" technique for changing a program's behavior.
- Describe the input-ignoring program in Figure 6.9.
- Show that YesOnString is recognizable.

Activity 10
- Construct a program that demonstrates that the language CrashOnString is recognizable.
- Show that if two languages are recognizable, then so is their intersection. Assume you have two programs that recognize the two languages, and built a program that recognizes their intersection.

Homework 6 6.3, 6.9, 6.10

Week 6 (10/08-10/12)

Day 1

- 7

Day 2

- 7, 8

Week 7 (10/15-10/19)

Day 1

- Fall Break

Day 2

- 8

Week 8 (10/22-10/26)

Day 1

- 9

Day 2

- 9

Week 9 (10/29-11/02)

Day 1

- 9

Day 2

- 10

Week 10 (11/05-11/09)

Day 1

- **Midterm 2**

Day 2

- 10, 11

Week 11 (11/12-11/16)

Day 1

- 11

Day 2

- 12

Week 12 (11/19-11/23)

Day 1

- 12

Day 2

- Thanksgiving Break

Week 13 (11/26-11/30)

Day 1

- 13

Day 2

- 13

Week 14 (12/03-12/07)

Day 1

- 14

Day 2

- 14